Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 197306, 11 pages
http://dx.doi.org/10.1155/2015/197306

Research Article

Adaptive CGFs Based on Grammatical Evolution

Jian Yao, Qiwang Huang, and Weiping Wang

College of Information System and Management, National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Jian Yao; markovyao@163.com

Received 29 July 2015; Revised 23 November 2015; Accepted 26 November 2015

Academic Editor: Andrzej Swierniak

Computer generated forces (CGFs) play blue or red units in military simulations for personnel training and weapon systems evaluation. Traditionally, CGFs are controlled through rule-based scripts, despite the doctrine-driven behavior of CGFs being rigid and predictable. Furthermore, CGFs are often tricked by trainees or fail to adapt to new situations (e.g., changes in battle field or update in weapon systems), and, in most cases, the subject matter experts (SMEs) review and redesign a large amount of CGF scripts for new scenarios or training tasks, which is both challenging and time-consuming. In an effort to overcome these limitations and move toward more true-to-life scenarios, a study using grammatical evolution (GE) to generate adaptive CGFs for air combat simulations has been conducted. Expert knowledge is encoded with modular behavior trees (BTs) for compatibility with the operators in genetic algorithm (GA). GE maps CGFs, represented with BTs to binary strings, and uses GA to evolve CGFs with performance feedback from the simulation. Beyond-visual-range air combat experiments between adaptive CGFs and nonadaptive baseline CGFs have been conducted to observe and study this evolutionary process. The experimental results show that the GE is an efficient framework to generate CGFs in BTs formalism and evolve CGFs via GA.

1. Introduction

Traditionally, computer generated forces (CGFs) in military simulation are controlled through rule-based scripts, such as simple rules, decision trees, and finite state machines (FSMs) [1]. However, these predefined rule-based CGFs lack the ability to adapt to changes in situations (e.g., an update of weapon systems or atypical enemy tactics) in analysis simulation [2]. CGFs are often tricked by trainees who quickly identify and exploit the weaknesses in unchanged scripts, resulting in degrading training effects [3]. In most cases, the SMEs rewrite behavior scripts to adapt CGFs to new scenarios or training tasks. It is costly and time-consuming to create, test, and maintain a large amount of scripts, especially in complex domain simulations, such as air combat [4].

The evolutionary process to adapt CGFs to complex military simulations is essentially Darwinian as the computation techniques are inspired by biological mechanisms, known as "survival of the fittest," providing a promising method to generate adaptive CGFs. Grammatical evolution (GE) is an evolutionary computation technique [5] which separates the search and solution spaces with user-specified grammar (usually grammar in Backus-Naur form). The domain knowledge is extracted and encoded with grammar to form phenotypic solutions. Through a genotype-to-phenotype mapping process based on the grammar, phenotypic solutions are transformed into a population of genotypic binary strings to evolve via genetic algorithm (GA) [6]. As GA is an independent evolutionary algorithm, SMEs focus their primary attention on encoding domain knowledge with this specified grammar.

Behavior trees (BTs) are a more modular, scalable, and reusable alternative to FSM in the development of Artificial Intelligence (AI) component [7]. The inherent modularity of BTs enables SMEs to decompose a complex task into simple tasks (subtrees) in a hierarchical way and build hierarchies (i.e., behavior trees) for different applications with low level task modules. All task modules are largely self-contained and have a common interface [8]; therefore, knowledge represented in BTs formalism is compatible with the genetic operators in GA. Further, as the task decomposition and action sequences are well organized in BTs formalism, CGFs models represented with BTs are easy for SMEs to read directly and validate with doctrine and experience.

In this paper, we introduce GE to generate adaptive CGFs, in which domain knowledge is encoded with BTs grammar in a modular and readable way, allowing both CGFs and domain knowledge to evolve with GA. To explore the benefits of our approach, experiments between adaptive CGFs and nonadaptive doctrine-driven CGFs are conducted to study this evolutionary process.

The outline of this paper is as follows: Section 2 is an overview of our proposed framework to generate adaptive CGFs with GE. Section 3 introduces BTs and details their specific application to air combat. Section 4 presents GE technique to evolve CGFs, followed by beyond-visual-range (BVR) air combat experiments to validate our method and also the analysis of the experimental results in Section 5. Conclusions and the future works are presented in Section 6.

2. Related Work

In most military simulation systems, CGFs are rule-based scripts (e.g., EADSIM [9], TAC BRAWLER [10], and JROADS [11]), which are often criticized for their inability to adapt to changes in the environment or deal with novel situations. Many different approaches have been introduced to improve the adaptivity of CGFs.

TacAir-Soar [12] adopted Soar Architecture [13] to encode intelligent behavior in air combat simulation. Although Soar is an effective production rule system to generate behavior, the static and nonchanging rule-based domain knowledge limits adaptivity. Toubman et al. applied dynamic scripting algorithm to generate behavior, which selected high weight rules from a rule base to build behavior scripts for combat simulation; then the results were fed back to update the weight of rules. This process continued until a reliable script is reached. Both single player and coordinate team behaviors created by dynamic scripting were tested in [14, 15]. Teng et al. proposed FALCON [16] architecture based on self-organizing neural network to evolve strategies in 1-v-1 dogfight training simulation. The proposed models showed significant improved adaptivity and higher performance in human-in-the-loop (HIL) experiments. The Smart Bandits project applied AI methods, such as cognitive modeling and machine learning, to generate human-like CGFs for air-to-air tactical training [17]. The main principle behind these systems is offline learning, adjusting the weights of rules with feedback from simulation. However, CGFs represented with weighted rule-sets models or neural network models were black box systems, from which it was difficult for SMEs to read the internal logics. As a result, SMEs could not produce a clear explanation of the relationship between behaviors and models. In practice, behaviors generated by these models are hard to analyze and validate [1].

Further research into building adaptive CGFs focused on mathematical models for decision-making. Virtanen et al. applied Game Theory to study optimal maneuvers in within-visual-range (WVR) air combat [18]. McGrew et al. proposed approximate dynamic programming (ADP) to compute near-optimal maneuvering decisions for Unmanned Aircraft System (UAS) [19]. However, due to the sheer number of variables, it is hard to build such a mathematic decision model that can consider all the aspects of air combat, including maneuver, fire control, and electric device operation, especially complex team tactics.

Recently, Air Force Research Lab (AFRL) Warfighter Readiness Research Division launched a Not-So-Grand Challenge (NSGC) project with industry teams to assess the capability of CGFs to adapt and exhibit realistic behavior, in an attempt to offer rapid adaptive realistic behavior modeling method for pilot training and rehearsal [20].

3. The Grammatical Evolution Approach

This section presents an overview of our proposed framework to generate adaptive CGFs with GE, including primary works to utilize GE and the process to evolve CGFs.

To utilize GE in air combat simulations, three primary works have to be undertaken. First, a syntax in Backus-Naur form (BNF) is defined, using BTs formalism to encode air combat domain knowledge. Then, a knowledge base (KB) is built with State-Action Space and classical tactics from both doctrine and expertise. Last, a mapping process between BNF grammar and integer strings is defined to run the GA. The details are presented in Sections 4 and 5.

The grammatical evolution approach to generating adaptive CGFs for air combat simulation proceeds as follows. First, SMEs build initial CGFs with modules in KB. Then, the CGFs are translated into integer strings to generate new CGFs with GA operators (inheritance, mutation, selection, and crossover). New CGFs are translated back into BTs as input to conduct air combat within simulation systems. The results of combat are measured to evaluate the fitness of CGFs, which are fed back to direct the GA operators. Figure 1 shows the evolutionary process of CGFs.

4. Behavior Trees for Domain Knowledge Representation

4.1. Behavior Trees. Behavior trees (BTs) originated from the computer game industry as a more modular, scalable, and reusable alternative to FSM in the development of Artificial Intelligence (AI) components [7]. Recently, the robotics community has shown great interest in BTs as a modular control formalism for UAVs and complex robots [21]. Colledanchise and Ögren provided a specific mathematical formula for BTs and further analyzed the safety, efficiency, and robustness of composite BTs [22].

A BT is a directed rooted tree defined as a two-tuple $BT = \langle V, E \rangle$, where

$V = A \cup C \cup N \cup \tau$ is the finite set of nodes with action nodes A, condition nodes C, control flow nodes N, and a root node τ;

$E \subset V \times V$ is a finite set of edges, $\forall \langle v_i, v_j \rangle \in E, v_i, v_j \in V$; v_i is parent node of v_j, while v_j is child of v_i.

The execution of a BT proceeds as follows. The root node sends signals called ticks to its children at a certain frequency. This tick is then passed down to leaf nodes (*Action* or *Condition*) with the guide of control flow nodes. Once a

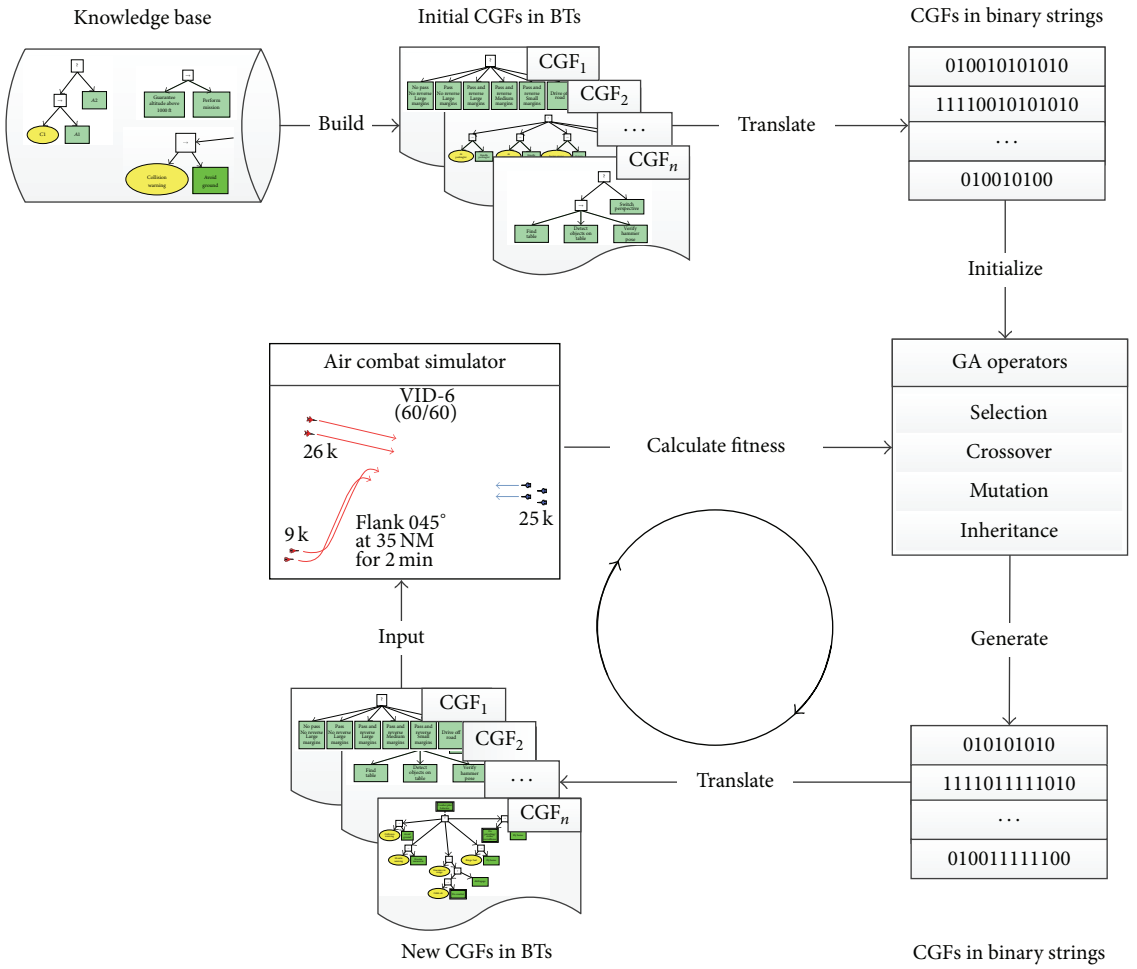


Figure 1: Process to generate adaptive CGFs.

leaf node receives a tick and executes its task, it returns to its parent status *Running* if its execution has not finished yet, *Success* if it has achieved its goal, or *Failure* otherwise, and *Condition* queries about the state and *Action* performs a specific task in the execution.

Control flow nodes are typically *Selector*, *Sequence*, *Parallel*, and *Decorator*; the first executes all its children from left to right until one fails, while the second executes its children until one succeeds. A *Parallel* node executes all its children sequentially, returns *Success* if a given number of children return *Success*, returns *Failure* if the remaining running children are not enough to reach the given number, even if they are all going to succeed, and returns *Running* otherwise. The *Decorator* sets constraints to pass ticks to its children. The algorithms of BTs nodes are presented in Algorithms 1–7. The symbols are S (*Success*), F (*Failure*), R (*Running*) $\subseteq X$, State Space $X(t) \in X$, and control signals $U(t) \in U$.

4.2. Encode Air Combat Domain Knowledge with BTs. Doctrine and expertise are the main source of domain knowledge in building CGFs for complex combat simulations. It ranges from low level device operations to high level tactics drawn from SMEs with real combat experience. In this research, we

```
for i ← 1 to N do
    state ← Tick(child(i));
    if state == Running then
        return Running;
    if state == Success then
        return Success;
return Failure;
```

Algorithm 1: Selector.

```
for i ← 1 to N do
    state ← Tick(child(i));
    if state == Running then
        return Running;
    if state == Failure then
        return Failure;
return Success;
```

Algorithm 2: Sequence.

```
for i ← 1 to N do
    state_i ← Tick(child(i));
if nSuccNodes(state_i) ≥ nS then
    return Success;
else if nFailNodes(state_i) ≥ nF
then
    return Failure;
else
    return Running;
```

Algorithm 3: Parallel.

```
if X_n(t) ∈ S_n then
    return Success;
if X_n(t) ∈ F_n then
    return Failure;
if X_n(t) ∈ R_n then
    U_n(t) ← γ_n(X_n(t));
return Running;
```

Algorithm 4: Action.

```
if X_n(t) ∈ S_n then
    return Success;
if X_n(t) ∈ F_n then
    return Failure;
```

Algorithm 5: Condition.

```
if Check(constrains) == True
then
    state ← Tick(child);
    return state;
else
    return Failure;
```

Algorithm 6: Decorator.

```
return Tick(child(0));
```

Algorithm 7: Root.

use the doctrine in [23], and the State and Action Space are as followed.

State Space. The chosen state variables are extracted from the cockpit instruments to make the simulation approximate to real-life exercises, such as the range and aspect angle of bandits displayed on radar, different alarming signals from Radar Warning Receiver (RWR), and cues of missile launch authority on Head-Up Display (HUD) calculated by fire control systems. The statuses of fighter are also considered, such as the speed, altitude, heading, and weapon accounts. The state variables are used to describe the condition nodes in BTs.

Action Space. The primary actions in air combat can be divided into maneuver, fire control, and device operation. As for BVR combat, the maneuver is much simpler than in WVR, where the pilot just sets the speed, heading, and altitude to approach of the adversary without performing a complex sequence of maneuvers. On the contrary, the operations on airborne devices and fire control systems are the core of BVR tactics. For example, while radar is the primary device to search for adversaries in battlefield, the emission of radar exposes the position and intention of pilot. Therefore, exactly when to activate radar is a critical decision in a sudden attack tactic. A total of 23 primary actions are provided in our simulation for CGFs.

Tactics. A tactic is a sequence of actions to achieve specific goals (e.g., missile evasion tactics), and most air combat tactics have been validated in training and real-life warfare. A tactic is represented as a subtree in KB, which is reusable as an independent block to build CGFs. Figure 2 describes a classical missile evasion tactic. A pilot makes a turn to place the enemy radar at 3 or 9 o'clock to break radar lock, activates an electronic countermeasure (ECM), and dispenses chaffs and decoys to jam missile seeker at the same time.

4.3. BNF Grammar for Air Combat. The BNF grammar in BTs formalism can be represented by a quaternary tuple:

$\text{BTs}_{\text{BNF}} = \langle T, G, S, P \rangle$, where

$T = A \cup C$ is the terminal set, including *Condition* C and *Action* A;

$G = N \cup \tau$ is the set of nonterminals, including control nodes N and root node τ;

$S = \tau$ is the start symbol (the root node in BTs);

P is a set of production rules.

The production rules in BNF grammar for air combat are defined as follows:

$\langle BT \rangle$ ::= $\langle BT \rangle \langle Node \rangle \mid \langle Node \rangle$

$\langle Node \rangle$ ::= $\langle Control \rangle \mid \langle Condition \rangle \mid \langle Action \rangle \mid \langle Root \rangle$

$\langle Control \rangle$ ::= *Selector* | *Sequence* | *Parallel* | $\langle Decorator \rangle$

$\langle Decorator \rangle$ ::= $\langle DecoratorType \rangle \langle Node \rangle$

$\langle DecoratorType \rangle$ ::= *Once* | *Repeat* | *Interval*10*s* | ⋯

$\langle Condition \rangle$ ::= $\langle State \rangle \mid \langle StateVariable \rangle \langle op \rangle (number)$

$\langle State \rangle$ ::= *RadarWarning* | *MissileWarning* | *LaunchAuthorized* | ⋯

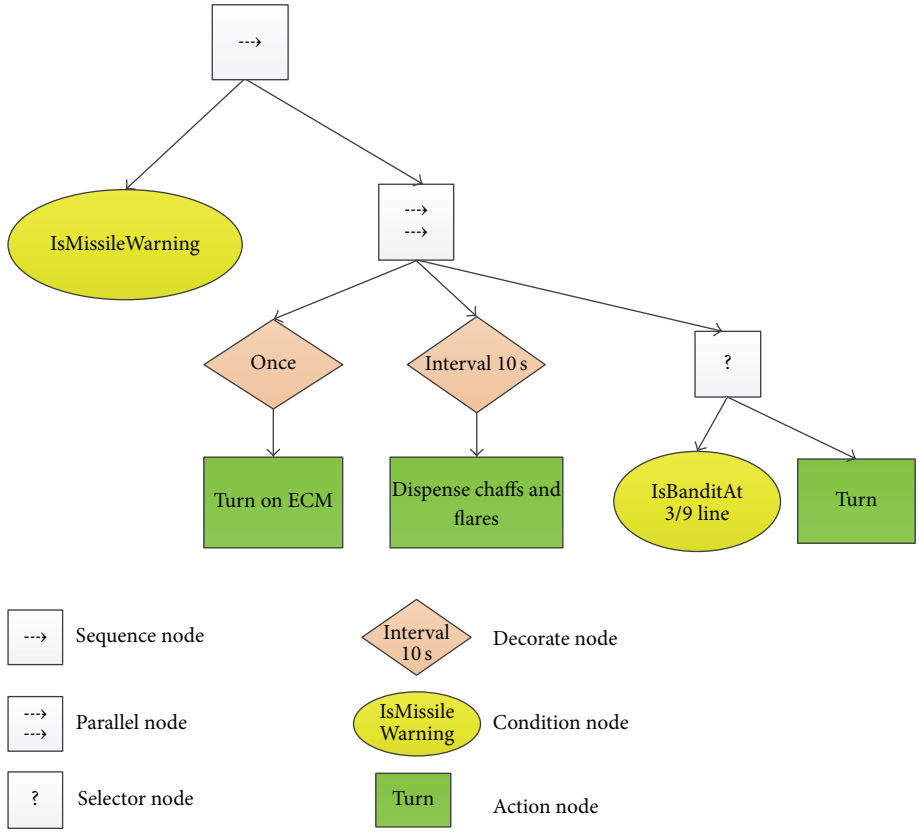


Figure 2: Example of missile evasion tactic behavior tree.

⟨*StateVariable*⟩ ::= *Altitude* | *Speed* | *Heading* | *TgtType* | *TgtHeading* | *TgtAltitude* | *TgtSpeed* | *TgtAspectAngle* | ⋯

⟨*op*⟩ ::== |<|>|<=|>=|

⟨*Action*⟩ ::= *SetSpeed* | *SetHeading* | *SetAlt* | *LockOn* | *Launch* | *ActivateECM* | *ActivateRadar* | ⋯

5. Grammatical Evolution for Adaptive CGF

5.1. Grammatical Evolution. Grammatical evolution is a grammar-based evolutionary computation technique pioneered by O'Neill and Ryan [5]. The core idea is to separate problem description from solution, in order to focus SMEs on the problem rather than the details in the search algorithm.

GE is composed of user-specified context-free grammar (usually grammar in BNF) to encode domain knowledge and an evolutionary algorithm (usually GA) to search for solutions; Figure 3 shows the components of grammatical evolution. GE takes a genotypic individual and maps it to a phenotype with BNF grammar. Fitness function evaluates the phenotype and genetic operators guide the evolution of the individual with fitness value.

5.2. Mapping Process between Integer Strings and BTs. BTs are the phenotype for CGF representation; to evolve CGFs with GA, BTs should be mapped to a genotype (e.g., integer or binary strings) for genetic operators.

The genotype-to-phenotype mapping proceeds as follows.

Integer strings (genotype) are used to choose production rules based on the mapping function

$$Rule = c \bmod r, \tag{1}$$

where c is the codon integer value in genotype and r is the number of rule choices for the current nonterminal symbol in BNF grammar. The process starts with an identified start symbol, in our case ⟨*BT*⟩. Then, the first element in genotype strings is used to select production rules. Figure 4 shows the mapping process of a missile launch tactic in BTs.

5.3. Fitness Function for CGFs. The fitness of CGFs is the guide for evolutionary process direction, just as with the object function in GA. While the final measure of performance for CGFs is *win or loss*, some intermediate measures can also be used to valuate the fitness. In our model, the performance of CGFs was measured with the following fitness function:

$$\text{Fitness} = w_1 * \text{Score} + w_2 * R_{\text{safe}} + w_3 * R_{\text{missile}}, \tag{2}$$

where Score is the result of the combat; the pilot is awarded 1 point for eliminating the enemy without loss, 1/3 for a draw,

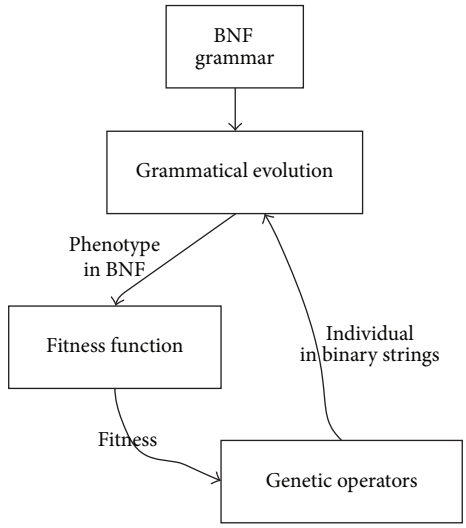
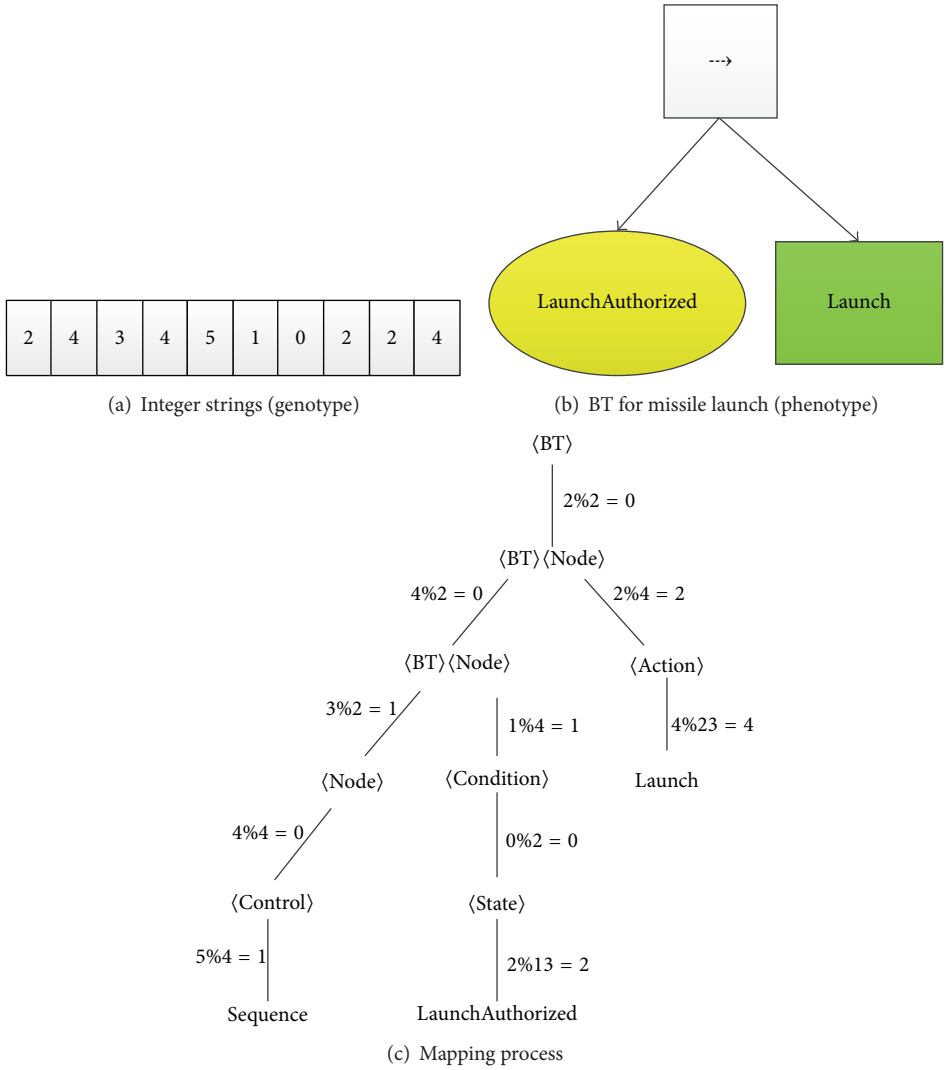


Figure 3: Components of grammatical evolution.



(a) Integer strings (genotype)

(b) BT for missile launch (phenotype)

(c) Mapping process

Figure 4: Mapping process from integer strings to missile launch behavior tree.

and 0 for a loss without eliminating the enemy. The R_{missile} is the ratio of missiles that hit target and were fired; R_{safe} is the ratio of safe time (not being locked by bandit radar or tracked by missiles) to the engagement time which begins once on-board devices have acquired bandit. w_i is the weight of measures, and $\sum_{i=1}^{3} w_i = 1$.

The three measures considered here reflect the main aspects of air combat, such as the final result of engagement, the resources consumed during the combat, and the risk to eliminate a bandit. The weights of the measure can be adjusted to achieve different types of CGF; for example, SMEs set a high w_2 to generate a more conservative pilot.

5.4. Evolution Strategy. While GA explores genotypic search space without limit, a large amount of syntactically correct phenotypic CGF solutions violates the doctrine and performance stupid behaviors. This is caused by "ripple effect" of GE: a small genotypic change can lead to a major phenotypic change. For example, a single gene mutation in chromosome causes a high level control flow node changing from Selector to Sequence; then the execution of BTs is totally different. To overcome this effect and maintain a stable evolutionary process, we employ two strategies to improve the efficiency of exploration.

First, we take a special grammar symbol $\langle XO \rangle$ to label crossover points for crossover operators [24]. As most subtactics (e.g., missile evasion) are self-contained blocks, we use $\langle XO \rangle$ to separate subtactics and indicate the crossover boundaries. During the evolution process, the context-sensitive tactics are exchanged as independent blocks. Both two-point crossover and one-point crossover are applied to improve the efficiency of exploration:

$$\begin{aligned}\langle CGF \rangle ::= \; & EvadeMissile = \langle BT \rangle; \langle XO \rangle; ApproachTgt = \\ & \langle BT \rangle; \langle XO \rangle; \\ & DogFight = \langle BT \rangle; \langle XO \rangle; GuideWeapon = \\ & \langle BT \rangle; \langle XO \rangle; \\ & DragAway = \langle BT \rangle; \langle XO \rangle; BreakLock = \\ & \langle BT \rangle; \langle XO \rangle;\end{aligned}$$

Second, we adopt a variable probability for mutation operator at different levels to keep the BTs structure stable. As CGFs behaviors are more sensitive to high level changes, we assign a lower mutation probability to high level control flow nodes. By contrast, low level nodes receive a relatively higher mutation probability. For N levels BTs, the top node (root) is level 0 and the lowest leaf node is level $N - 1$. Assuming P_m is the mutation probability parameter for GA, we set $P_m(i) = P_m * (1 - e^{-i})$ as the mutation probability for nodes at level i.

6. Case Study

6.1. Scenario and Simulation. As an exploratory study, we tested our method in a 1-v-1 BVR air combat scenario. Both red and blue were equipped with identical fighters, missiles, and avionic systems. The initial height was set at 9000 m with a velocity of 0.9 Ma for both sides and the distance between them established as 450 km. Both sides received an initial position and velocity of adversary from Air Early Warning (AEW) at the beginning of the task. The experiments were conducted with the Weapon Effectiveness Simulation System (WESS), a simulation platform for equipment effectiveness analysis [25]. Figure 5 shows a screenshot of the scenario in simulation.

Table 1: Parameters for genetic algorithm.

Parameter	Value
Population_size	100
Generations	100
Elite_size	15
Tournament_size	4
Single-point crossover ratio	50%
Two-point crossover ratio	50%
Mutation probability	5%

6.2. Experimental Setup. The objective of experiments was to evaluate the effectiveness of adaptive CGFs in an air combat scenario. The red pilot was a nonadaptive doctrine-driven CGF set at Expert level. The blue pilot was assigned an identical fighter and weapons and used adaptive CGFs to engage red. Two different levels, Trainee and Expert, were set as the initial level for blue pilot at the beginning of the evolutionary process. The weights in fitness function were $W = [0.7, 0.1, 0.2]$ for both sides. The parameters for GE are given in Table 1. It is noteworthy that the population size is 100, which means that we should initialize 100 CGFs. As creating such a large number of CGFs is quite laborious, we instead created only 20 CGFs for each level and generated another 80 initial CGFs with mutations and crossovers. In each generation, every blue CGF in the population fought 10 times against the red adversary, and a total of $100 * 100 * 10$ simulations were conducted.

6.3. Results. The performance of adaptive CGFs in each generation was measured with the average fitness value of the population, as well as the overall Exchange Ratio (ER), defined as the ratio of the number of red losses divided by blue losses over a series of encounters. Figure 6 shows the evolutionary process of average fitness of adaptive Blue Trainee and adaptive Blue Expert in the engagement with Red Expert. The average fitness value of adaptive Blue Trainees started with 0.1 and quickly rose to about 0.32 after 70 generations and then stagnated across the subsequent 30 generations. Conversely, the fitness of Red Expert decreased from 0.62 to 0.42. As for the adaptive Blue Expert, the initial fitness is 0.29, below Red Expert; the reason was that 80% of initial Blue Experts were generated randomly. However, the average fitness of blue increased to the same level of red, at about 0.4.

Figure 7 shows the evolutionary process of the ER in each generation. Figure 7(a) shows an increase in the ER as the adaptive Blue Trainees evolved over generations, with ER from about 10% to 40%, and the top 15% fitness populations even reaching 50% by generation 60, meaning that the elites of blue achieved equal capabilities to red after sufficient

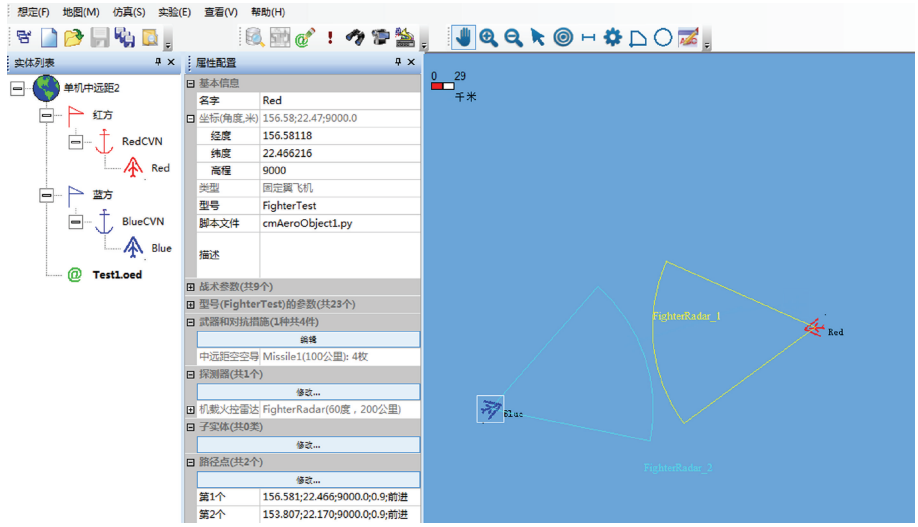
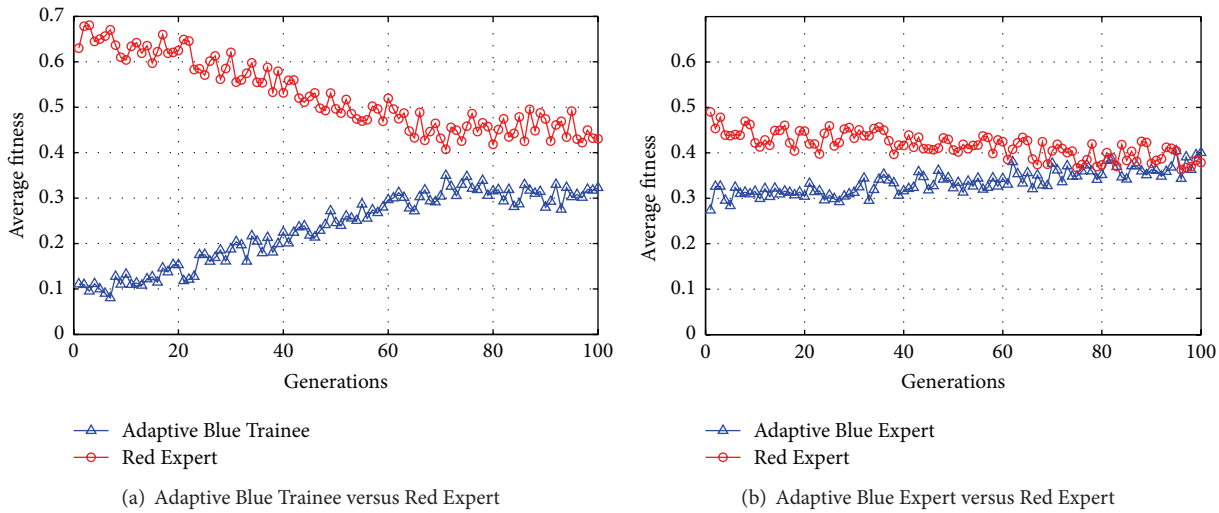


Figure 5: Screenshot of 1-v-1 BVR air combat scenario.



(a) Adaptive Blue Trainee versus Red Expert

(b) Adaptive Blue Expert versus Red Expert

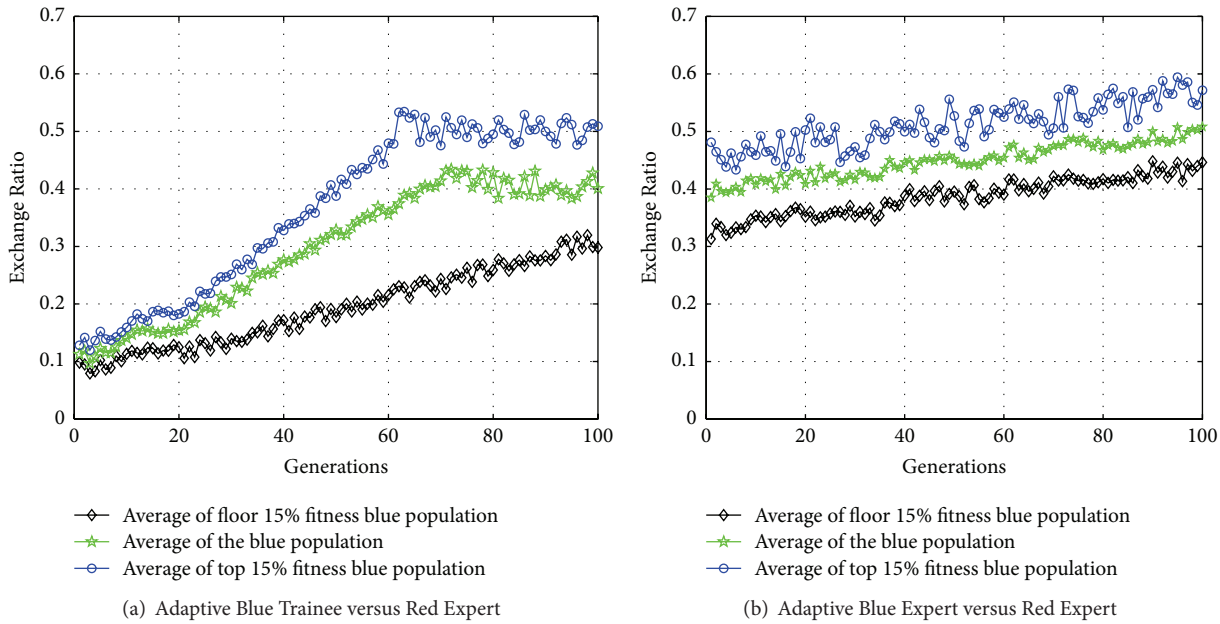
Figure 6: Fitness of adaptive blue CGFs in the evolution.

evolution. Figure 7(b) shows the result of combat between adaptive Blue Expert and nonadaptive Red Expert. Due to the random initializations, the ER of blue started slightly lower than 50%, before scoring a draw at 50%. The top 15% fitness of blue outperformed red with ER above 50% after 60 generations and t-test at the 5% significance level with a 50% ER, for the last 60 generations were unable to reject the null hypothesis that average ER is greater than 50%. At the same time, the fluctuation of the top 15% population shows the variety of generated elite CGFs.

6.4. Validation of Adaptive CGFs. Validation, in simple terms, is the process of determining the degree to which a model meets the requirements of the simulation [26]. In this particular instance, we focused on the adaptivity of CGFs during simulation and data drawn from the experiment results supported our hypothetical function of the models. Consequently, the objective of validation here was to check whether the adaptive CGFs violated the doctrine and expertise.

In actuality, to date, the most common means of validating CGFs' behaviors has been through face validation [1], meaning that SMEs watch the combat animation or analyze the logs of simulation. During the preliminary face validation, five SMEs independently validated 10% of the top 15% fitness CGFs in the last 10 generations and did not observe any obvious behaviors that violated the doctrine. However, the so-called "stupid" behaviors were commonly identified in the low fitness value CGFs.

Furthermore, a direct method was proposed to validate CGFs based on the readability of BTs formalism. Since task decomposition and action sequences were well organized



(a) Adaptive Blue Trainee versus Red Expert

(b) Adaptive Blue Expert versus Red Expert

Figure 7: Exchange Ratio of adaptive blue CGFs in the evolution.

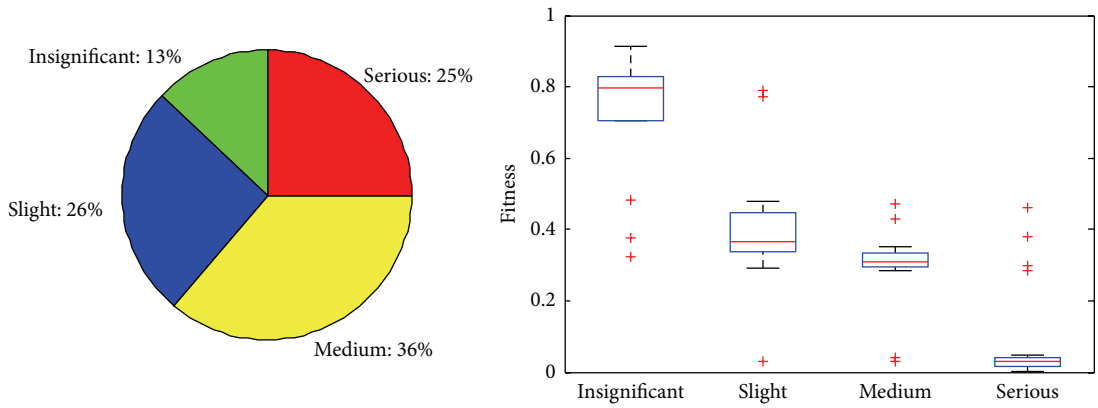


Figure 8: Statistical results of direct validation on CGFs.

in BTs, the SMEs were able to directly read and analyze CGFs models rather than rely on face validation. Without knowledge of the combat results, SMEs examined 100 CGF samples from evolution process and evaluated their degrees of violating doctrine and expertise. The degrees were classified as "serious," "medium," "slight," and "insignificant."

The statistical results of direct validation are shown in Figure 8. The proportions of each degree are presented in the pie chart: "insignificant" level CGFs amounts to 13%, "serious" and "slight" account for one-quarter each, and the largest part is "medium" CGFs with 36%. The attached boxplot compares the fitness value to the degree of violating doctrine and expertise. The fitness and degree were almost consistent in the whole, and CGFs that seriously violated doctrine scored the lowest fitness, about 0.05, while CGFs without significant errors got medium fitness of 0.79. The difference between CGFs in groups "medium" and "slight" was not significant; medium fitness was 0.37 and 0.31, respectively. The outliers in each group were worthy of note in that SMEs' judgments mismatched combat performances. The reason for these inconsistencies might be the lack of expertise or the novel knowledge emerging in the evolutionary process.

According to the results of both face validation and direct validation by SMEs, we came to the conclusion that adaptive CGFs with high fitness were consistent with doctrine and expertise and fitness function was an effective method to ensure the validity of CGFs.

7. Conclusions and Future Work

In this paper, we have presented a novel framework to generate adaptive CGFs based on grammatical evolution, which encode domain knowledge in modular behavior trees and evolve CGFs via GA. Adaptive CGFs for air combat were

developed and tested in BVR engagement with nonadaptive doctrine-driven CGFs. The experiments demonstrated the adaptivity of generated CGFs with increasing fitness value and an overall Exchange Ratio.

Future work will focus on the verification and validation of the generated CGFs and analysis of the variety in evolution. Mining innovative tactic from GE generated CGFs will also be a new challenge, and related works are being conducted via machine learning techniques. We also plan to apply these methods to human-in-the-loop (HIL) training and study the evolutionary process when training with real-life pilots.

Conflict of Interests

The authors declare no conflict of interests.

Authors' Contribution

Drafting of paper was performed by Jian Yao. Model construction was done by Jian Yao and Qiwang Huang. Simulation experiments were conducted by Jian Yao. Planning and supervision of the research were performed by Weiping Wang.

Acknowledgment

This work is partly supported by the National Natural Science Foundation of China (no. 61273198 and no. 91324014).

References

[1] NATO, "Human behavior representation in constructive simulation," Tech. Rep. RTO-TR-HFM-128, NATO RTO, 2009.

[2] S. J. E. Taylor, A. Khan, K. L. Morse, A. Tolk, L. Yilmaz, and J. Zander, "Grand challenges on the theory of modeling and simulation," in *Proceedings of the Symposium on Theory of Modeling & Simulation—DEVS Integrative M&S Symposium (DEVS '13)*, p. 34, San Diego, Calif, USA, April 2013.

[3] S. Mittal, M. J. Doyle, and A. M. Portrey, "Human in the loop in System of Systems (SoS) modeling and simulation," in *Modeling and Simulation Support for System of Systems Engineering Applications*, chapter 16, pp. 415–451, John Wiley & Sons, 2014.

[4] S. Mittal, M. J. Doyle, and E. Watz, "Detecting intelligent agent behavior with environment abstraction in complex air combat systems," in *Proceedings of the 7th Annual IEEE International Systems Conference (SysCon '13)*, pp. 662–670, IEEE, Orlando, Fla, USA, April 2013.

[5] M. O'Neill and C. Ryan, *Grammatical Evolution*, Springer, New York, NY, USA, 2003.

[6] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT press, 1998.

[7] A. Champandard, "Behavior trees for next-gen game AI," in *Proceedings of the Game Developers Conference (GDC '07)*, Audio Lecture, December 2007.

[8] I. Millington and J. Funge, *Artificial Intelligence for Games*, CRC Press, 2012.

[9] Teledyne Brown Engineering Incorporated, *EADSIM Executive Summary*, Teledyne Brown Engineering Incorporated, 2012.

[10] R. Kerchner, "The TAC BRAWLER air combat simulation analyst manual (rev. 3.0)," Decision Science Application Report 668, 1985.

[11] P. Kerbusch and P. Eigeman, "Flexible and reusable tactical behaviour models for combat aircraft," Technical Report, DTIC Document, 2010.

[12] J. E. Laird, K. J. Coulter, O. M. Jones, P. G. Kenny, F. Koss, and P. E. Nielsen, "Integrating intelligent computer generated forces in distributed simulation: TacAir-Soar," in *Proceedings of the Simulation Interoperability Workshop (STOW '98)*, Orlando, Fla, USA, March 1998.

[13] J. Laird, *The Soar Cognitive Architecture*, MIT Press, 2012.

[14] A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat, and J. van den Herik, "Dynamic scripting with team coordination in air combat simulation," in *Modern Advances in Applied Intelligence: 27th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2014, Kaohsiung, Taiwan, June 3–6, 2014, Proceedings, Part I*, vol. 8481 of *Lecture Notes in Computer Science*, pp. 440–449, Springer, 2014.

[15] A. Toubman, J. J. Roessingh, P. Spronck, A. Plaat, and J. van den Herik, "Improving air-to-air combat behavior through transparent machine learning," in *Proceedings of the Interservice/Industry Training, Simulation & Education Conference (I/ITSEC '14)*, Orlando, Fla, USA, 2014.

[16] T.-H. Teng, A.-H. Tan, Y.-S. Tan, and A. Yeo, "Self-organizing neural networks for learning air combat maneuvers," in *Proceedings of the Annual International Joint Conference on Neural Networks (IJCNN '12)*, pp. 1–8, IEEE, Brisbane, Australia, June 2012.

[17] J. Roessingh, R. Rijken, R. Merk et al., "Modelling CGFs for tactical air-to-air combat training motivation-based behaviour and machine learning in a common architecture," Tech. Rep. NLR-TP-2011-540, National Aerospace Laboratory NLR, 2011.

[18] K. Virtanen, J. Karelahti, and T. Raivio, "Modeling air combat by a moving horizon influence diagram game," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 5, pp. 1080–1091, 2006.

[19] J. S. McGrew, J. P. How, B. Williams, and N. Roy, "Air-combat strategy using approximate dynamic programming," *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 5, pp. 1641–1654, 2010.

[20] M. J. Doyle and A. M. Portrey, "Rapid adaptive realistic behavior modeling is viable for use in training," in *Proceedings of the 23rd Conference on Behavior Representation in Modeling and Simulation (BRIMS '14)*, pp. 73–80, Washington, DC, USA, April 2014.

[21] O. Petter, "Increasing modularity of UAV control systems using computer game behavior trees," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, AIAA 2012-4458, Minneapolis, Minn, USA, August 2012.

[22] M. Colledanchise and P. Ögren, "How Behavior Trees modularize robustness and safety in hybrid systems," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '14)*, pp. 1482–1488, Chicago, Ill, USA, September 2014.

[23] R. L. Shaw, *Fighter Combat: Tactics and Maneuvering*, Naval Institute Press, 1985.

[24] M. Nicolau and I. Dempsey, "Introducing grammar based extensions for grammatical evolution," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '06)*, pp. 648–655, July 2006.

[25] Y.-L. Lei, Q. Li, F. Yang, W.-P. Wang, and Y.-F. Zhu, "A composable modeling framework for weapon systems effectiveness simulation," *System Engineering Theory & Practice*, vol. 33, no. 11, pp. 2954–2966, 2013.

[26] DMSO, "VV & A recommended practices guide," Tech. Rep., Defense Modeling and Simulation Office, 2015, http://www.msco.mil/VVA_RPG.html.